A Methodology for Interrupt Analysis in Virtual Platforms

Puneet Dhar
Agenda

• Introduction
• Interrupts in Virtual Prototyping
• Challenges in Debugging Interrupts
• Proposed Analysis Methodology
• RH850 Adaptation
• Conclusions
Pre-Requisites

- Basic understanding of
  - System C scheduling
  - Temporal Decoupling
  - Interrupt Handling in Instruction Set Simulators (ISS)
Introduction

• An interrupt is a request to the processor to suspend its current program and transfer control to the Interrupt Service Routine (ISR).

• A developer must know if the design is
  – Processor intensive
  – Interrupt intensive

• Interrupt handling should be
  – Timely (esp. when safety critical)
  – Memory and processor efficient
Interrupts in Virtual Prototyping

• A key usecase of any Virtual Prototype (VP) is system software development
• Use of Interrupts is common in system software
• Important for developers to understand interrupts and prevent associated issues – interrupt overloading, stack overflows etc.
• Empirically it's known, interrupts
  – Hard to get completely right
  – Difficult to track down

With right means, VP can be a powerful tool to design efficient software for interrupt based systems
Challenges in Debugging Interrupts

- Peculiar/rare preconditions
- Multiple sources and sinks
- Interrupt Enable/Disable
- Masks/Priority settings
- Missed Interrupts
- Separate Execution Flow
- Nested Interrupts, Reentrancy
- Context management/Stack Corruption
- Multi-level ISR, Delayed Procedural Calls (DPC)
- Unnoticed Faults/Violations/Traps
Additional challenges in a VP

If the VP uses temporarily decoupled ISS:

• Interrupts are only handled at quantum boundaries.
• Interrupts external to VP (e.g. pulse from USB) can be missed
• A large static quantum leads to
  – Loss of Inter Processor Communication
  – Missed Peripheral Interrupts
  – Timing Issues
Shortcomings in existing methodology

• Investigate various traces
  – Debug Log trace
  – Instruction, function, register trace
  – Pin Value trace

• Verify correct execution by
  – Assembly level debugging
  – Putting breakpoints on pins/registers/instructions
  – Examining values in pins, registers

• Manual, time and effort intensive
  – Sparsely distributed information
  – Time to converge to actual issue may take multiple days
A solution helps if it

- Is easy to integrate with an ISS
- Meaningfully presents the distributed information on
  - Events, register settings and actual handling
  - Latencies and execution of ISR
  - Missed/Pending/Masked interrupts
- Helps developer assess if it’s an interrupt related problem
- Helps developer to profile the software
Proposed Interrupt Analyzer based Methodology
A Quick Glance

ISS

IssIf

Analyzer Pulls Information

ISS Pushes Event

Interrupt Analyzer

AnalyzerIf

Information Flow

DB

© Accellera Systems Initiative
How to integrate an ISS

Implement an IssIf to

– Register interrupt capabilities
  • HW, SW Interrupts
  • No. of channels for HW
  • HW IRQ pins
  • HW ACK pins
  • ISR Addresses
– Allow Analyzer to fetch
  • Core Time
  • Register Values
  • Program Counter
  • Pin Values

Use AnalyzerIf to

– Signal Pin Events:
  • IRQ
  • ACK
– Signal ISR Event:
  • Request check/Notify for ISR entry
  • Notify returns from ISR
Implementation Overview

Analyzer Interface (PUSH Interface)
- Push Interrupt Request
- Push Acknowledgement
- Request ISR Entry Check
- Push ISR Exit

Analysis ISS Interface (PULL Interface)
- Pull Initial information on
  - Interrupt Names
  - HW/SW type
  - Channel Nos.
  - Register Set
  - ISR Address Strategy
- Pull information at relevant PUSH Events
  - Core Time
  - Program Counter Info
  - Register Values
  - Pin Values
## Output: Informatics

<table>
<thead>
<tr>
<th>Analysis Elements</th>
<th>Details covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Events</td>
<td>IRQ, ACK, Messages</td>
</tr>
<tr>
<td>ISR Events</td>
<td>Entry, Exit, Fall-Through</td>
</tr>
<tr>
<td>Latencies</td>
<td>ACK Response, ISR Duration (DPC not covered)</td>
</tr>
<tr>
<td>IRQ Special Events</td>
<td>Masked IRQ, Pending IRQ, Missed IRQ</td>
</tr>
<tr>
<td>Register Set</td>
<td>Mask, Status and Cause registers</td>
</tr>
</tbody>
</table>
Timeline Chart

Interrupt states
- Interrupt Request (e.g. For RH850 FEINT, EIINT)
- Acknowledgement (ACK)
- No Hardware Interrupt(NO_HW_INT)

CPU States
- NORMAL Execution
- ISR
- ISR_EXIT
- ISR_FALLTHROUGH

Pin Events
## Information Table

**Example 1 from Use Cases**

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>CoreTime</th>
<th>FEI</th>
<th>FEP</th>
<th>FEPSW</th>
<th>ISRTT</th>
<th>LastPC PC</th>
<th>PinName</th>
<th>RT</th>
<th>Count</th>
<th>% Duration</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Execution</td>
<td>NORMAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.06%</td>
<td></td>
</tr>
<tr>
<td>Interrupt Events</td>
<td>NO_HW_INT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>5.87%</td>
<td></td>
</tr>
<tr>
<td>Interrupt Events</td>
<td>FEINT</td>
<td>330000</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>6698</td>
<td>66994</td>
<td>FEINTRQ</td>
<td>0</td>
<td>0.18%</td>
<td></td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>340000</td>
<td>240</td>
<td>66996</td>
<td>196616</td>
<td>0</td>
<td>66996</td>
<td>240</td>
<td>0</td>
<td>0</td>
<td>51.74%</td>
<td></td>
</tr>
<tr>
<td>Interrupt Events</td>
<td>ACK</td>
<td>340000</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>66994</td>
<td>66996</td>
<td>FEINTACK</td>
<td>10000</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>Interrupt Events</td>
<td>NO_HW_INT</td>
<td>340000</td>
<td>0</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>66994</td>
<td>66996</td>
<td>FEINTACK</td>
<td>0</td>
<td>93.76%</td>
<td></td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>3160000</td>
<td>240</td>
<td>66996</td>
<td>196616</td>
<td>0</td>
<td>2820000</td>
<td>250</td>
<td>66992</td>
<td>0</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>CPU Execution</td>
<td>NORMAL</td>
<td>3160000</td>
<td>240</td>
<td>66996</td>
<td>196616</td>
<td>0</td>
<td>250</td>
<td>66992</td>
<td>0</td>
<td>0</td>
<td>42.02%</td>
<td></td>
</tr>
</tbody>
</table>

- **CoreTime**
- **Registers**
- **ISR Duration computed at ISR Exit**
- **Response Time at Acknowledgment**
- **Utilization**
RH850 Based Adaption

- A platform with Renesas RH850 ISS interfaced with an interrupt controller and peripherals
- Quantum: Both dynamic and static configurations
- Virtual Platform is designed running in Synopsys Virtualizer environment
Platform in use
Example Use Cases
(1) Analyzing a HW Interrupt:

- Normal Execution
- ISR Execution Starts
- ISR Exit
- Interrupt Request
- Acknowledgment
(2) Analyzing Nested Interrupts

- ISR Starts
- Nested ISR Execution
- ISR Fall Through
- ISR Exit
- Last ISR Exit

Normal Execution

© Accellera Systems Initiative
(2) Analyzing Nested Interrupts

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>CoreTime</th>
<th>ISRName</th>
<th>ISR Turnaround Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>30400000</td>
<td>EIINT5</td>
<td>25650000</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>30170000</td>
<td>EIINT4</td>
<td>21120000</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>29940000</td>
<td>EIINT3</td>
<td>16690000</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>29710000</td>
<td>EIINT2</td>
<td>12360000</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>29480000</td>
<td>EIINT1</td>
<td>8130000</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR_EXIT</td>
<td>29250000</td>
<td>EIINT1</td>
<td>4000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>CoreTime</th>
<th>PinName</th>
<th>ResponseTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIINT</td>
<td>ACK</td>
<td>25250000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
<tr>
<td>EIINT1</td>
<td>ACK</td>
<td>21350000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
<tr>
<td>EIINT2</td>
<td>ACK</td>
<td>17350000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
<tr>
<td>EIINT3</td>
<td>ACK</td>
<td>13250000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
<tr>
<td>EIINT4</td>
<td>ACK</td>
<td>9050000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
<tr>
<td>EIINT5</td>
<td>ACK</td>
<td>4750000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
<tr>
<td>EIINT6</td>
<td>ACK</td>
<td>350000</td>
<td>EIINTACK</td>
<td>10000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object</th>
<th>State</th>
<th>CoreTime</th>
<th>ISRName</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>350000</td>
<td>EIINT5</td>
<td>14.18%</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>475000</td>
<td>EIINT5</td>
<td>13.86%</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>905000</td>
<td>EIINT4</td>
<td>13.54%</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>1325000</td>
<td>EIINT3</td>
<td>13.21%</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>1735000</td>
<td>EIINT2</td>
<td>12.89%</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>21350000</td>
<td>EIINT1</td>
<td>12.57%</td>
</tr>
<tr>
<td>CPU Execution</td>
<td>ISR</td>
<td>25250000</td>
<td>EIINT</td>
<td>12.89%</td>
</tr>
</tbody>
</table>
(3) Analyzing Missed Interrupts

INT_0 ACK

INT_1 (a)

INT_1 (b),
(a) was not serviced

INT_0 ACK

INT_1(b) is
taken

INT_0 (High Priority)

INT_1

MISSED

INT_0 ISR

INT_0 ISR

INT_0 ISR

INT_1(b) ISR taken
(4) Analyzing Pending Interrupts

Interrupt event (EINT)
Interrupt event (FEINT)
Interrupt event (ACK)
Interrupt event (NO_HW_INT)
CPU execution (ISR)
CPU execution (NORMAL)

INT_0

INT_1

Interrupt assertion activates a HW timer.

ISR Starts, disables interrupts. HW timer stopped

INT_1 assertion reactivates HW timer

Timer expires, terminates simulation

Interrupt Disabled => Interrupt Pending

Timer expires, terminates simulation

Interrupt assertion activates a HW timer.
Conclusions

Analyzer consolidates the distributed debugging techniques and helps

• Organize interrupt data in tables
• Graphically assess interrupt handling
• Point out interrupt anomalies
• Track software exceptions
• Expedite debugging and profiling
• Expedite setting of optimal static quantum
• Present a post-silicon use case for debugging interrupt issues
Questions